小马哥的 Java 项目实战营

Java EE 项目 - 第五节 配置管理与 Java Logging

小马哥（mercyblitz）

我是谁？

小马哥（mercyblitz）

- 父亲
- Java 劝退师
- Apache Dubbo PMC
- Spring Cloud Alibaba 架构师
- 《Spring Boot 编程思想》作者

议题

- 配置管理
- Java Logging
- 问答互动

配置管理

- 简介

配置（Configuration）是应用程序重要的元信息（Metadata），几乎所有软件依赖配置调控程序行为，比如 MySQL 中的 my.ini 文件，Apache Maven 的配置文件 settings.xml 文件，以及其项目管理文件 pom.xml。尽管配置常以文件的形式承载，然而并仅限于此。根据配置所处的物理位置，可分为本地配置（Local Configuration）和远程配置（Remote Configuration）。其中，本地配置相对于程序的位置主要包括“内部配置（Internal Configuration）”以及“外部配置（External Configuration）”，而远程配置则有“版本化配置（Versioned Configuration）”和“分布式配置（Distributed Configuration）”等实现。

配置管理

- 本地配置（Local Configuration）

顾名思义，本地配置即配置存放在应用程序所在的物理环境，比如程序内部、物理机器或虚拟化容器。存放在程序进程内部的配置称之为“内部配置（Internal Configuration）”，相反则是“外部配置（External Configuration）”。

配置管理

- 内部配置（Internal Configuration）

通常，内部配置是通过程序代码，甚至是硬编码（Hard Code）实现。虽然不推荐程序使用硬编码来初始化配置，然而这并非是不可原谅的做法，如：

```
public Customizer<SentinelCircuitBreakerFactory> defaultConfig() {
    return factory -> {
        factory.configureDefault(
            id -> new SentinelConfigBuilder().resourceName(id)
            .rules(Collections.singletonList(new DegradeRule(id)
                                             .setGrade(RuleConstant.DEGRADE_GRADE_RT)
                                             .setCount(100)
                                             .setTimeWindow(10)))
            .build());
    };
}
```

配置管理

- 内部配置（Internal Configuration）

内部配置并未与硬编码（Hard-Code）划上等号，如 Java 8 引入的共享线程池 java.util.concurrent.ForkJoinPool#commonPool()：

```
private static ForkJoinPool makeCommonPool() {

    final ForkJoinWorkerThreadFactory commonPoolForkJoinWorkerThreadFactory =
        new CommonPoolForkJoinWorkerThreadFactory();
    int parallelism = -1;
    ......
        if (parallelism < 0 && // default 1 less than #cores
            (parallelism = Runtime.getRuntime().availableProcessors() - 1) <= 0)
            parallelism = 1;
    if (parallelism > MAX_CAP)
        parallelism = MAX_CAP;
    return new ForkJoinPool(parallelism, factory, handler, LIFO_QUEUE,
                            "ForkJoinPool.commonPool-worker-");
}
```

配置管理

- 外部配置（External Configuration）

外部配置是软件使用者、开发者以及运维者最常见和熟悉的配置手段，正如前文提到的 MySQL my.ini 文件等。在 Java 生态体系中，JDK 层面允许程序读取来自 Java System Properties、操作系统环境变量以及 JNDI 等多方配置来源。

```
private static ForkJoinPool makeCommonPool() {

    final ForkJoinWorkerThreadFactory commonPoolForkJoinWorkerThreadFactory =
            new CommonPoolForkJoinWorkerThreadFactory();
    int parallelism = -1;
    try {  // ignore exceptions in accessing/parsing properties
        String pp = System.getProperty
            ("java.util.concurrent.ForkJoinPool.common.parallelism");
        ......
        if (pp != null)
            parallelism = Integer.parseInt(pp);
```

配置管理

- 远程配置（Remote Configuration）

在分布式场景中，远程配置的内容来自于配置运用端程序进程物理环境以外的环境，其作用相当重要，是大多数互联网企业的基础设施标配，大多数实现采用 C/S（客户端/服务器）架构。C/S 架构广义地包含了 B/S（浏览器/服务器）架构，这两种实现均采用 B/S 架构，属于“分布式配置（Distributed Configuration）B/S 架构，然而都归类于“分布式配置实现，即应用需要依赖一个专属的 HTTP 客户端，通过访问远端的 Web 服务器获取配置。实现上，配置客户端版本化配置（Versioned Configuration）”。

配置管理

- Java 标准外部化配置
  - Java SE
    - Java 系统属性 - java.lang.System#getProperties()
    - 操作系统环境变量 - java.lang.System#getenv()
    - 偏好配置 - java.util.prefs.Preferences
  - Java EE
    - Servlet 上下文配置 - javax.servlet.ServletContext#getInitParameter(String)
    - Servlet 配置 - javax.servlet.ServletConfig#getInitParameter(String)
    - JSP 配置 - javax.servlet.descriptor.JspConfigDescriptor
    - JNDI 配置 - javax.naming.Context#lookup(javax.naming.Name)

配置管理

- Apache 通用配置开源框架 - Apache Commons Configuration
  - 支持数据源
    - Properties files
    - XML documents
    - Windows INI files
    - Property list files (plist)
    - JNDI
    - JDBC Datasource
    - System properties
    - Applet parameters
    - Servlet parameters

配置管理

- Apache Commons Configuration 1.x - 配置源
  - 接口信息
    - PropertiesConfiguration Loads configuration values from a properties file
    - XMLConfiguration Takes values from an XML document
    - INIConfiguration Loads the values from a .ini file as used by Windows
    - PropertyListConfiguration Loads values from an OpenStep .plist file
    - JNDIConfiguration Using a key in the JNDI tree, can retrieve values
    - BaseConfiguration An in-memory method of populating a Configuration object
    - HierarchicalConfiguration An in-memory Configuration object
    - SystemConfiguration A configuration using the system properties

配置管理

- Apache Commons Configuration 1.x - 数据类型
  - BigDecimal
  - BigInteger
  - boolean
  - byte
  - double
  - float
  - int
  - long
  - short
  - String

配置管理

- Apache Commons Configuration 1.x - 操作方法
  - addProperty()
    - Adds a new property to the configuration. If this property already exists, another value is added to it (so it becomes a multi-valued property)
  - clearProperty()
    - Removes the specified property from the configuration
  - setProperty()
    - Overwrites the value of the specified property. This is the same as removing the property and then calling addProperty() with the new property value.
  - clear()
    - Wipes out the whole configuration

配置管理

- Apache Commons Configuration 1.x - 使用
  - Maven 坐标

```
<dependency>
    <groupId>commons-configuration</groupId>
    <artifactId>commons-configuration</artifactId>
    <version>1.10</version>
</dependency>
```

  - 依赖
    - http://commons.apache.org/proper/commons-configuration/dependencies_1_10.html [点击访问]

Java Logging

- 简介

Java日志API，在包中介绍java.util.logging文件，通过生成适合最终用户、系统管理员、现场服务工程师和软件开发团队分析的日志报告，促进客户站点的软件服务和维护。日志api捕获应用程序或平台中的安全故障、配置错误、性能瓶颈和/或bug等信息。

核心包支持将纯文本或XML格式的日志记录传递到内存、输出流、控制台、文件和套接字。此外，日志API能够与主机操作系统上已经存在的日志服务进行交互。

Java Logging

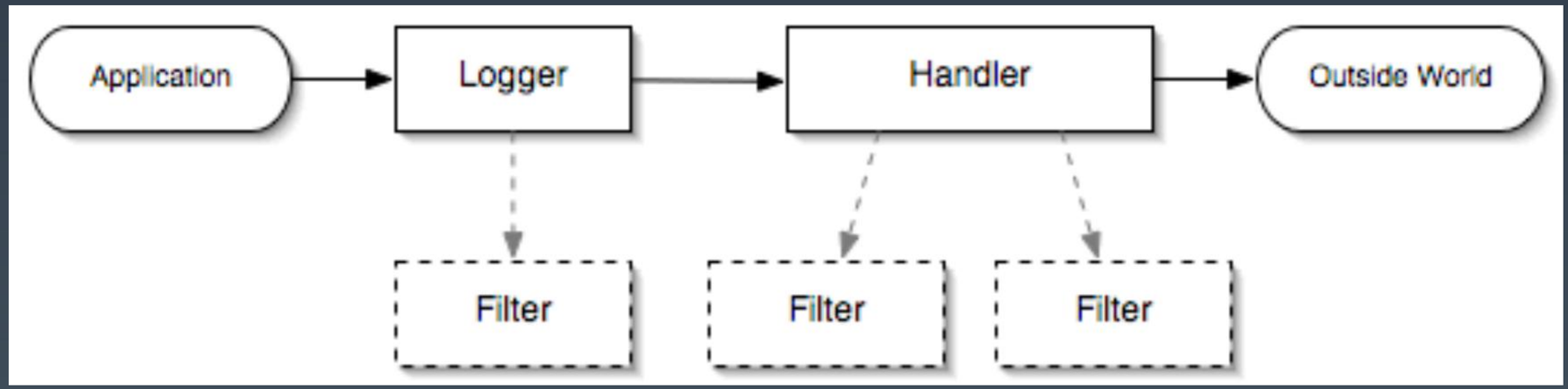
- 控制流程

应用程序对 Logger 对象进行日志记录调用。记录器被组织在一个层次化的名称空间中，子记录器可以从名称空间中的父记录器继承一些日志属性。

应用程序对 Logger 对象进行日志记录调用。这些 Logger 对象分配传递给处理程序对象以供发布的 LogRecord 对象。 Loggers 和 Handlers 都可以使用日志级别（Levels）和（可选） Filters 来决定是否对特定的 LogRecord 感兴趣。当需要在外部发布 LogRecord 时，Handlers 可以（可选）使用 Formatter 在将消息发布到 I/O 流之前对其进行本地化和格式化。

Java Logging

- 控制流程



Java Logging

- 日志级别（Levels）

每个日志消息都有一个关联的日志级别。该级别粗略地说明了日志消息的重要性和紧迫性。日志级对象封装了一个整数值，值越大表示优先级越高。

Level类定义了七个标准日志级别，从FINEST（最低优先级，具有最低值）到SEVERE（最高优先级，具有最高值）。

API - java.util.logging.Level

Java Logging

- 日志级别（Levels）

每个日志消息都有一个关联的日志级别。该级别粗略地说明了日志消息的重要性和紧迫性。日志级对象封装了一个整数值，值越大表示优先级越高。

Level类定义了七个标准日志级别，从FINEST（最低优先级，具有最低值）到SEVERE（最高优先级，具有最高值）。

API - java.util.logging.Level

Java Logging

- 日志对象（Loggers）

客户机代码向 Logger 对象发送日志请求。每个 Logger 都跟踪它感兴趣的日志级别，并丢弃低于此级别的日志请求。

Logger 通常使用 "." 分隔名称（如"java.awt". 名称空间是分层的，由 LogManager管理。名称空间通常应与Java打包名称空间保持一致，但不需要盲目跟随。

Logger 在记录名称空间中记录其父 Logger。Logger 的父级是其在记录名称空间中最接近的现有祖先。根日志程式（名称为""）没有父级。匿名日志记录者都被授予根日志记录者作为其父。

Java Logging

- 日志对象（Loggers）

  Logger 可从其在记录者名称空间中的父类继承各种属性。特别是， Logger 可继承：

  - Levels - 如果一Logger的level设置为空，则该Logger将使用一个有效的level，该level将通过走上父树并使用第一个非空level获得。

  - Handlers - 默认情况下， Logger 会将任何输出消息记录到其父级的 Handlers 中，依此在树中循环上升。

  - ResourceBundle 名称 - 如果 Logger 的资源包名称为空，则其将继承为其父级定义的任何资源包名称，依此在树上循环。

Java Logging

- 日志处理器（Handlers）

  Java SE 提供以下日志处理器实现：

  - StreamHandler - 一个简单的处理程序，用于将格式化记录写入OutputStream
  - ConsoleHandler - 向其写入格式记录的简单处理程序System.err
  - FileHandler - 将已格式化的日志记录写入一个文件或一组循环日志文件的处理程序
  - SocketHandler - 将已格式化的日志记录写入远程TCP端口的处理程序
  - MemoryHandler - 在内存中缓冲日志记录的处理程序

Java Logging

- 日志格式化器（Formatters）

  Java SE 提供两种 Formatter 实现：

  - SimpleFormatter - 编写简要的“人类可读”的日志记录摘要

  - XMLFormatter - 编写详细的XML结构信息

Java Logging

- 日志管理器（LogManager）

全局的 LogManager 对象，用于跟踪全局日志记录信息。LogManager 对象可以使用静态 LogManager.getLogManager 方法获得。在 LogManager 初始化期间根据系统属性创建的。此属性允许容器应用程序（如EJB容器）用其自身的LogManager子类代替默认类。

LogManager 对象包括：

- 命名的 Logger 分层名称空间
- 从配置文件中读取的一组日志记录控制属性

Java Logging

- 日志配置文件（Configuration File）

在启动时读取的日志配置文件来初始化日志配置。此日志配置文件为标准 java.util.Properties 格式。或者，可以通过指定可用于读取初始化属性的类来初始化日志记录配置。此机制允许从任意源（如LDAP、JDBC等）读取配置数据。有关详细信息，请参阅LogManager API规范。

有一小部分全局配置信息。这在 LogManager 类的描述中指定，并包括在启动期间安装的根级别处理程序列表。

初始配置可指定特定记录者的级别。这些级别应用于命名层次结构中的命名日志记录程序及其下的任何日志记录程序。级别按在配置文件中定义的顺序应用。

Java Logging

- 日志默认配置（Default Configuration）

JRE附带的默认日志记录配置仅为一个默认值，并可由ISV、系统管理员和最终用户覆盖

默认配置仅使用有限的磁盘空间。它不会向用户大量提供信息，但确保始终获取关键故障信息

默认配置在根日志记录程序上建立一个处理程序，用于向控制台发送输出

Java Logging

- 日志动态配置更新（Dynamic Configuration Updates）

  程序能够在运行期动态地更新以下配置：

  - FileHandlers, MemoryHandlers, 和 ConsoleHandlers 动态创建和属性更新
  - Handlers 能被动态地添加或移除
  - 新的 Logger 对象能被动态地创建，并能关联指定的 Handlers
  - 特定 Handlers 上的级别能被动态地设置

Java Logging

- 日志本地化（Localization）

  日志消息可能需要本地化。

每个 Logger 可能有一个与其相关联的资源包名称。相应的资源包可用于在原始消息串和本地化消息串之间进行映射。

通常，本地化将由 Formatter 执行。为方便起见， Formatter 类提供了一个提供一些基本本地化和格式设置支持的 Format 消息方法。

THANKS!